How the properties of applicants affect whether they are eligible for credit card application

Haizhou Li (Undergraduate)

2020-12

Risk control method in the financial industry is a really important. We can use personal information and data from credit card applicants to predict the probability of future defaults and credit card borrowings. The bank is able to decide whether to issue a credit card to the applicant. Credit scores can objectively quantify the magnitude of risk.

Generally speaking, credit score cards are based on historical data. Once encountered huge economic fluctuations. Past models may lose their original predictive power. Logical model is a common method of credit scoring. Because Logistic is suitable for binary classification tasks and can calculate the coefficient of each feature. In order to facilitate understanding and operation, the scorecard will multiply the logistic regression coefficient by a certain value (such as 100) and round it up.

At present, with the development of machine learning algorithms. More predictive methods have been introduced in credit card scoring, such as Boosting, Random Forest and Support Vector Machines. However, these methods generally do not have good transparency. It may be difficult to provide customers and regulators with reasons for rejection or acceptance.

Building a machine learning model to predict whether an applicant is a "good" or "bad" customer, unlike other tasks, does not give a definition of "good" or "bad". You should use some techniques, such as year analysis, to construct labels. In addition, the problem of unbalanced data is also a big problem in this task

In [50]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
import statsmodels.api as sm
from string import ascii_letters
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import plotly.express as px
import cufflinks as cf
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn import metrics
```

In [3]:
```
application = pd.read_csv('application_record.csv')

credit = pd.read_csv('credit_record.csv')
credit.head()
```

Out[3]:

	ID	MONTHS_BALANCE	STATUS
0	5001711	0	X
1	5001711	-1	0
2	5001711	-2	0
3	5001711	-3	0
4	5001712	0	C

Months_Balance: The month of the extracted data is the starting point, backwards, 0 is the current month, -1 is the previous month, and so on

Data Cleaning

Group by the ID in credit data, since the applicants in the original data may have several balances but only use one ID number. Then add up the Months_Balance numbers of several accounts from one user

In [4]:
```
begin_month = pd.DataFrame(credit.groupby(["ID"])["MONTHS_BALANCE"].agg(min
# Merge it to the original dataframe
application = pd.merge(application, begin_month,how="left",on="ID")
```

Find out the duplicated ID numbers and then remove these rows.

In [5]:
```
drop_index = application[application.duplicated(subset=['ID'], keep=False)]

application_new = application.drop(drop_index)
```

Then combine the application data and the credit data

In [6]:
```
data_new = application_new.merge(begin_month, on='ID', how='left')
```

In [7]:
```
data_new.head()
```

Out[7]:

	ID	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_
0	5008804	M	Y	Y	0	4
1	5008805	M	Y	Y	0	4
2	5008806	M	Y	Y	0	1
3	5008808	F	N	Y	0	2
4	5008809	F	N	Y	0	2

Now I'd like to start a method to define whether the applicant is high risk applican by classify the Months_Balance and if the history of the balance is long enough, we could say that this applicant is relatively low risk.

In [8]:
```
def risk(x):
    if x >= -3:
        return 'no'
    elif x < -3:
        return 'yes'
    else:
        return 'null'

data_new['RISK'] = data_new['MONTHS_BALANCE_x'].apply(lambda x: risk(x))
```

In [9]:
```
data_new.drop(data_new[data_new['RISK'] == 'null'].index, inplace=True)
```

LabelEncoder is a utility class to help normalize labels such that they contain only values between 0 and n_classes-1. This is sometimes useful for writing efficient Cython routines.

In [10]:
```
le = LabelEncoder()
data_new['MALE'] = le.fit_transform(data_new['CODE_GENDER'])
data_new['CAR'] = le.fit_transform(data_new['FLAG_OWN_CAR'])
data_new['REALTY'] = le.fit_transform(data_new['FLAG_OWN_REALTY'])

data_new.drop(['CODE_GENDER','FLAG_OWN_CAR','FLAG_OWN_REALTY'], axis=1, inp
```

In [11]:
```
data_new['AGE'] = data_new['DAYS_BIRTH'].apply(lambda x: round(abs(x/365)))
data_new['YEARS_EMPLOYED'] = data_new['DAYS_EMPLOYED'].apply(lambda x: roun

data_new.drop(['DAYS_BIRTH','DAYS_EMPLOYED'], axis=1, inplace=True)
```

In [12]:
```
data_new['INCOME'] = data_new['AMT_INCOME_TOTAL'].apply(lambda x: x/1000)

data_new.drop(['AMT_INCOME_TOTAL'], axis=1, inplace=True)
```

In [13]:
```
data_new.head()
```

Out[13]:

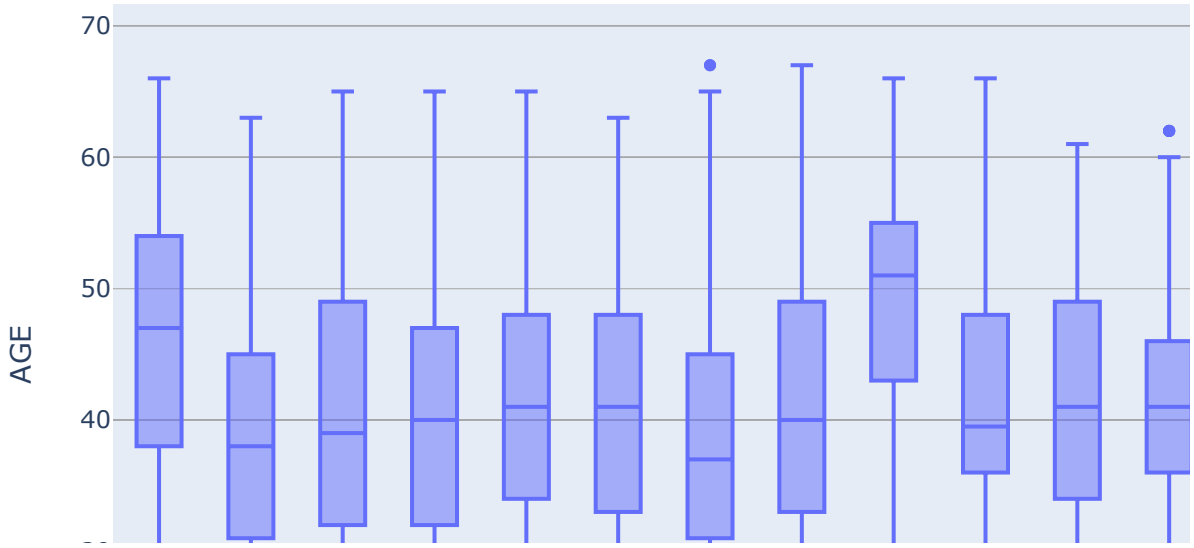
	ID	CNT_CHILDREN	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATU
0	5008804	0	Working	Higher education	Civil marriag
1	5008805	0	Working	Higher education	Civil marriag
2	5008806	0	Working	Secondary / secondary special	Marrie
3	5008808	0	Commercial associate	Secondary / secondary special	Single / not marrie
4	5008809	0	Commercial associate	Secondary / secondary special	Single / not marrie

5 rows × 21 columns

Data Visualization

Visualization the data by using Plotly Express in Python. The plotly.express module (usually imported as px) contains functions that can create entire figures at once, and is referred to as Plotly Express or PX. Plotly Express is a built-in part of the plotly library, and is the recommended starting point for creating most common figures.

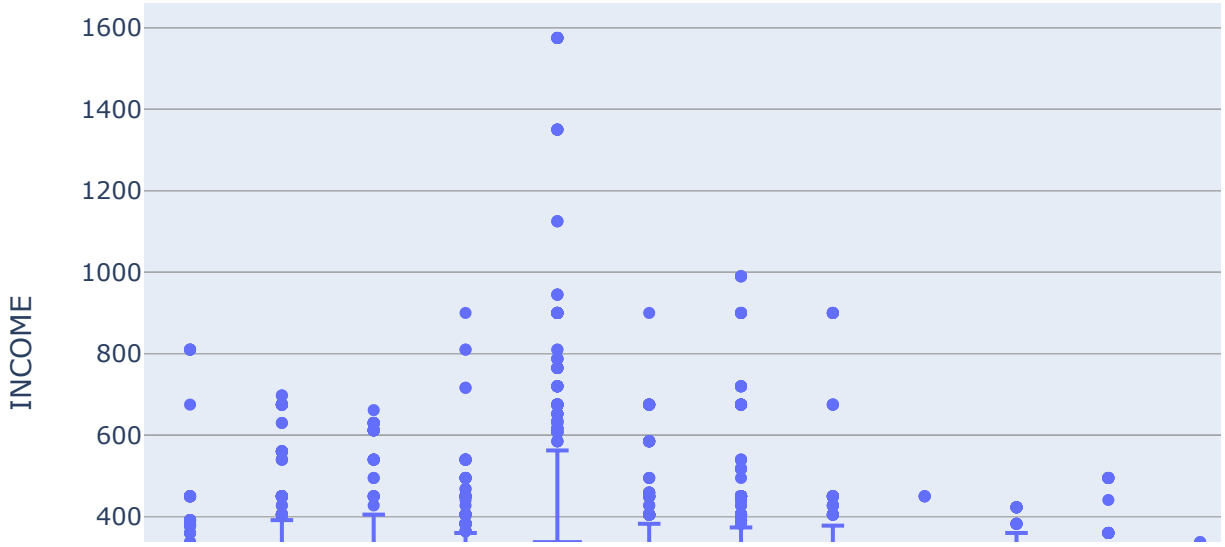
In [14]:
```
px.box(data_frame=data_new, x='OCCUPATION_TYPE', y='AGE')
```



Normally, if the population is in a huge amount, then the median can reflect the actual the most of people's level. Cleaning staff and security staff are obviously the higher aged workers.

We can choose another average level which is Laborers to compare with Security Staff

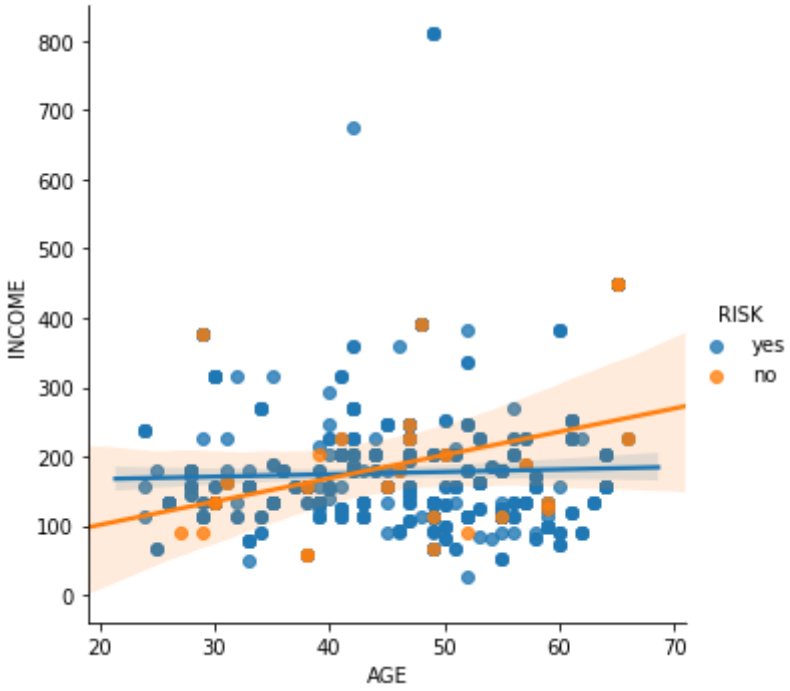
In [15]:
```
px.box(data_frame=data_new, x='OCCUPATION_TYPE', y='INCOME')
```



Then the scatter plot could show more relationships between age income and risk

In [16]:
```
sns.lmplot(data=data_new[data_new['OCCUPATION_TYPE']=='Security staff'], x=
```

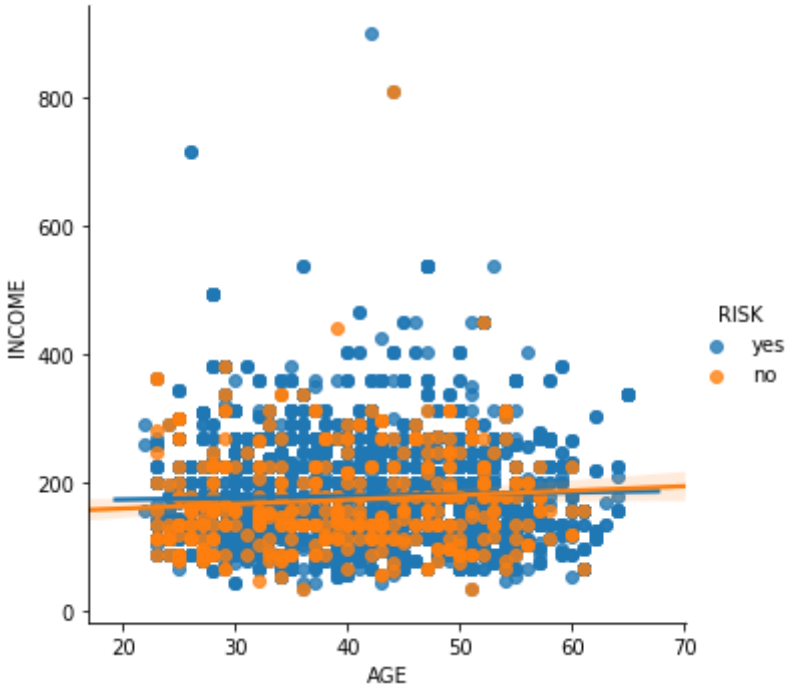
Out[16]: <seaborn.axisgrid.FacetGrid at 0x7f800c279978>



For the security staff, the less risk applicants have a trend of increasing wage along the age increasing.

In [17]:
```
sns.lmplot(data=data_new[data_new['OCCUPATION_TYPE']=='Laborers'], x='AGE',
```

Out[17]:
```
<seaborn.axisgrid.FacetGrid at 0x7f800c75f2e8>
```



Now we want to delete Laborers having (age > than 45 and Salary >200K) and INCOME > 400¶

In [18]:
```
#Deleting Laborers having (age > than 45 and Salary >200K) and INCOME > 400
data_new.drop(data_new.query('OCCUPATION_TYPE=="Laborers" and AGE > 45 and

data_new.drop(data_new.query('OCCUPATION_TYPE=="Laborers" and INCOME > 400'
```

Now we want to delete Security Staff having (age > than 50 and Salary >200K) and INCOME > 400¶

In [19]:
```
#Deleting outliers for Security staff
data_new.drop(data_new.query('OCCUPATION_TYPE=="Security staff" and AGE > 5

data_new.drop(data_new.query('OCCUPATION_TYPE=="Security staff" and INCOME
```

Now make the Risk colunm become a dummy variable

In [24]:
```
data_new['RISK'] = le.fit_transform(data_new['RISK'])
```

In [25]:
```
data_new['MARRIED'] = data_new['NAME_FAMILY_STATUS'].apply(lambda x: 1 if (
```

In [26]:
```
DEGREE = pd.get_dummies(data_new['NAME_EDUCATION_TYPE'], drop_first=True)
OCCUPATION = pd.get_dummies(data_new['NAME_INCOME_TYPE'], drop_first=True)
```

In [27]:
```
data_new = pd.concat([data_new, DEGREE, OCCUPATION], axis=1)
```

In [28]:
```
data_new.head()
```

Out[28]:

	ID	CNT_CHILDREN	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATU
0	5008804	0	Working	Higher education	Civil marriag
1	5008805	0	Working	Higher education	Civil marriag
2	5008806	0	Working	Secondary / secondary special	Marrie
3	5008808	0	Commercial associate	Secondary / secondary special	Single / not marrie
4	5008809	0	Commercial associate	Secondary / secondary special	Single / not marrie

5 rows × 30 columns

Then we can group by the Incom Type, Educational Level, and Occupation Type

In [29]:
```
_occp = data_new.groupby(['NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','OCCUPAT

lay(data_occp)
```

	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	OCCUPATION_TYPE	count
0	Commercial associate	Academic degree	Managers	2
1	Commercial associate	Academic degree	Sales staff	12
2	Commercial associate	Higher education	Accountants	293
3	Commercial associate	Higher education	Cleaning staff	11
4	Commercial associate	Higher education	Cooking staff	14
5	Commercial associate	Higher education	Core staff	384
6	Commercial associate	Higher education	Drivers	103
7	Commercial associate	Higher education	HR staff	22
8	Commercial associate	Higher education	High skill tech staff	157
9	Commercial associate	Higher education	IT staff	14
10	Commercial associate	Higher education	Laborers	192

In [30]:
```
data_occp = data_occp[data_occp.groupby(['NAME_INCOME_TYPE','NAME_EDUCATION

data_occp
```

Out[30]:

	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	OCCUPATION_TYPE	count
1	Commercial associate	Academic degree	Sales staff	12
12	Commercial associate	Higher education	Managers	696
25	Commercial associate	Incomplete higher	Managers	74
33	Commercial associate	Lower secondary	Laborers	16
43	Commercial associate	Secondary / secondary special	Laborers	1016
53	Pensioner	Higher education	Accountants	1
54	Pensioner	Higher education	Core staff	1
55	Pensioner	Higher education	Laborers	1
56	Pensioner	Higher education	Managers	1
57	Pensioner	Higher education	Medicine staff	1
58	Pensioner	Secondary / secondary special	Core staff	4

Since we only want the float or int value to do the regression, we need to remove the null values and change the strings to dummy variable and fill in 1 or 0.

In [33]:
```
def occp_clean(values):
    profession = values[0]
    degree = values[1]
    occupation = values[2]
    if pd.isnull(occupation):
        for index,row in data_occp.iterrows():
            if ((row['NAME_INCOME_TYPE'] == profession) and (row['NAME_EDUC
                return row['OCCUPATION_TYPE']
    else:
        return occupation


data_new['OCCUPATION'] = data_new[['NAME_INCOME_TYPE','NAME_EDUCATION_TYPE'
```

In [34]:
```
Occu = pd.get_dummies(data_new['OCCUPATION'], drop_first=True)

Occu.head()
```

Out[34]:

	Cleaning staff	Cooking staff	Core staff	Drivers	HR staff	High skill tech staff	IT staff	Laborers	Low-skill Laborers	Managers	Medicine staff	
0	0	0	0	0	0	0	0	0	0	1	0	
1	0	0	0	0	0	0	0	0	0	1	0	
2	0	0	0	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	

Since we don't need that many variables and now we want to shrink the variables to 3 and combine some similar variables, by according to the previous visulized box plot data.

'Cleaning staff','Cooking staff','Drivers','Laborers','Low-skill Laborers','Security staff','Waiters/barmen staff' are considerd as level 3 workers

'HR staff','Sales staff','Secretaries','Medicine staff','Private service staff' are considered as level 2 workers according to the previous graaph

'Managers','Core staff','High skill tech staff','IT staff','Realty agents' are level 1 workers represent the highest wage jobs

In [35]:
```
Level3 Workers'] = Occu[['Cleaning staff','Cooking staff','Drivers','Laborer
Level2 Workers'] = Occu[['HR staff','Sales staff','Secretaries','Medicine st
Level1 Workers'] = Occu[['Managers','Core staff','High skill tech staff','I


rop(['Cleaning staff','Cooking staff','Drivers','Laborers','Low-skill Labore
```

In [36]:
```
Occu.head()
```

Out[36]:

	Level3 Workers	Level2 Workers	Level1 Workers
0	0	0	1
1	0	0	1
2	1	0	0
3	0	1	0
4	0	1	0

In [37]:
```
data_new = pd.concat([data_new, Occu], axis=1)
```

In [38]:
```
data_new.head()
```

Out[38]:

	ID	CNT_CHILDREN	NAME_INCOME_TYPE	NAME_EDUCATION_TYPE	NAME_FAMILY_STATU
0	5008804	0	Working	Higher education	Civil marriag
1	5008805	0	Working	Higher education	Civil marriag
2	5008806	0	Working	Secondary / secondary special	Marrie
3	5008808	0	Commercial associate	Secondary / secondary special	Single / not marrie
4	5008809	0	Commercial associate	Secondary / secondary special	Single / not marrie

5 rows × 34 columns

In [39]:
```
data_new['OWN_HOUSE'] = data_new['NAME_HOUSING_TYPE'].apply(lambda x: 1 if
```

Clean up the redondent data and only keep numerical data

In [40]:
```
data_new.drop(['ID','NAME_INCOME_TYPE','NAME_EDUCATION_TYPE','NAME_FAMILY_S

data_new.rename(columns={'CNT_CHILDREN':'CHILD','FLAG_MOBIL':'MOBIL','FLAG_
data_new.head()
```

Out[40]:

	CHILD	MOBIL	FLAG_WORK_PHONE	PHONE	EMAIL	FAMILY_MEMBERS	MONTHS_BALANCE_y
0	0	1	1	0	0	2.0	-15.0
1	0	1	1	0	0	2.0	-14.0
2	0	1	0	0	0	2.0	-29.0
3	0	1	0	1	1	1.0	-4.0
4	0	1	0	1	1	1.0	-26.0

5 rows × 27 columns

Split data into a training set and test set.

In [41]:
```
X = data_new.drop('RISK', axis=1)
y = data_new['RISK']


X_bal,y_bal = SMOTE().fit_sample(X,y)

X_bal = pd.DataFrame(X_bal, columns=X.columns)
```

In [42]:
```
X_train, X_test, y_train, y_test = train_test_split(X_bal,y_bal,stratify =
```

Methodology

I decide to use Logist Regression which is the same asmodelling the probability of getting a loan from a bank

The more reasons as follow: Binary logistic regression requires the dependent variable to be binary.

For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.

Only the meaningful variables should be included.

The independent variables should be independent of each other. That is, the model should have little or no multicollinearity.

The independent variables are linearly related to the log odds.

Logistic regression requires quite large sample sizes.

In [47]:
```
logReg = LogisticRegression()

logReg.fit(X_train, y_train)

y_predict = logReg.predict(X_test)
```

```
/Users/haizhouli/anaconda3/lib/python3.7/site-packages/sklearn/linear_mod
el/logistic.py:432: FutureWarning:

Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to si
lence this warning.
```

In [48]:
```
y_predict
```

Out[48]:
```
array([1, 1, 1, ..., 1, 1, 0])
```

In [45]:
```
print("Train Accuracy")
print(logReg.score(X_train, y_train))
print("\nTest Accuracy")
print(logReg.score(X_test, y_test))

print("\nThe Area Under the ROC curve")
print(roc_auc_score(y_test, y_pred))

print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Train Accuracy
1.0

Test Accuracy
1.0

The Area Under the ROC curve
1.0
[[10019     0]
 [    0 10018]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     10019
           1       1.00      1.00      1.00     10018

    accuracy                           1.00     20037
   macro avg       1.00      1.00      1.00     20037
weighted avg       1.00      1.00      1.00     20037
```

The Area Under the ROC curve (AUC) is an aggregated metric that evaluates how well a logistic regression model classifies positive and negative outcomes at all possible cutoffs. It can range from 0.5 to 1, and the larger it is the better. People will sometimes use the AUC as a means for evaluating predictive performance of a model, although because it represents all possible cutoff values, which isn’t feasible in practice, the interpretation is difficult. We recommend interpreting the ROC curve directly as a way to choose a cutoff value. Therefore, the ROC and accuracy is accurately reflected the model's reliablility

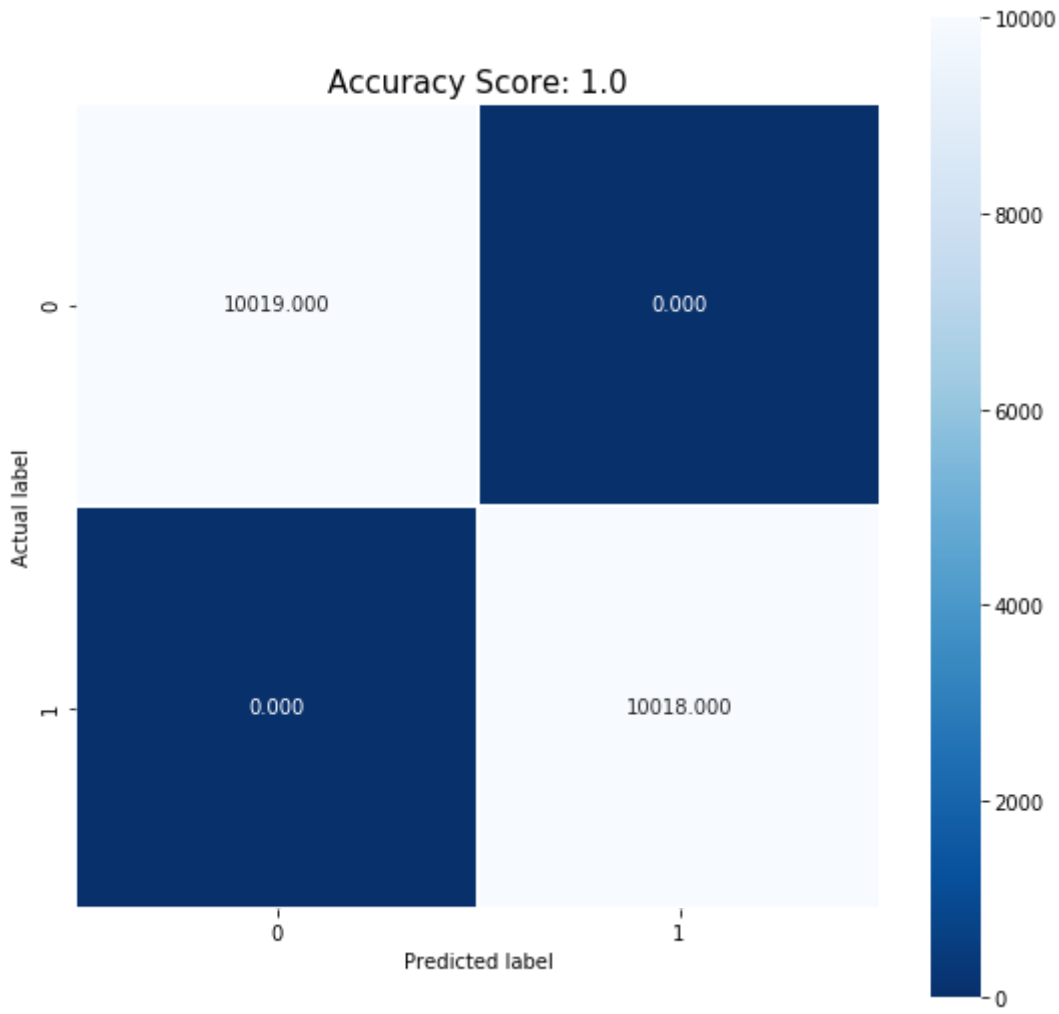
The train accuracy: The accuracy of a model on examples it was constructed on.

The test accuracy is the accuracy of a model on examples it hasn't seen.

Confusion matrix: A tabulation of the predicted class (usually vertically) against the actual class (thus horizontally).

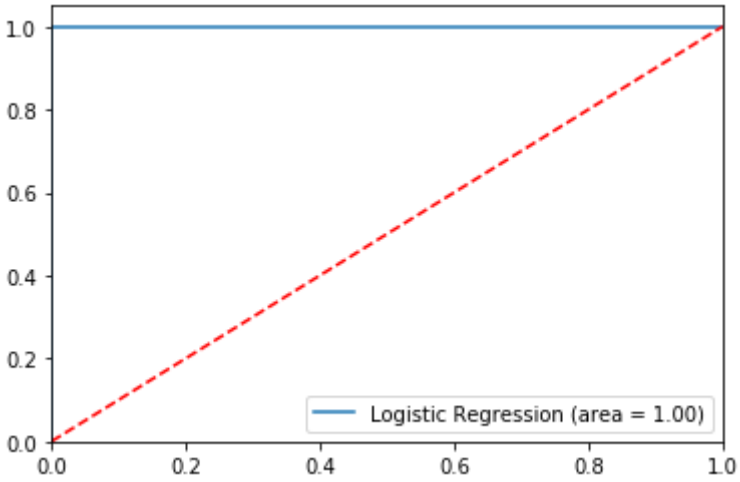
this method produces a more understandable and visually readable confusion matrix using seaborn.

In [54]:
```
cm = metrics.confusion_matrix(y_test, y_predict)
score = logReg.score(X_test, y_test)
plt.figure(figsize=(9,9))
sns.heatmap(cm, annot=True, fmt=".3f", linewidths=.5, square = True, cmap =
plt.ylabel('Actual label');
plt.xlabel('Predicted label');
all_sample_title = 'Accuracy Score: {0}'.format(score)
plt.title(all_sample_title, size = 15);
```



In [58]:
```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, logReg.predict(X_test))
fpr, tpr, thresholds = roc_curve(y_test, logReg.predict_proba(X_test)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_a
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.legend(loc="lower right")
```

Out[58]: <matplotlib.legend.Legend at 0x7f8013c8dc50>



This model building actually cost lots of time and the result is 1.0 which is not a convincable conclution, after the deadline of this research, I will keep doing on it and find better approches and also I will try Gradient Descent to optimize the data set.

In [ ]: